# TASK DESCRIPTION

This is basically a tax calculator program where program reads tax values from text file and applies data for tax calculation. After calculating tax amount, program then, stores all the data in another tax report file. Whole program is completed by following steps:

1. Welcome message is displayed first to start the tax calculator program.
2. It checks if both the files exist or not. If not then new files with data and its specific format are created.
3. If file exist, then it reads data from both the files and stores in appropriate structure to use later for calculation.
4. Then program asks three options from like to calculate tax for employee or to search existed data.
5. For option 1, program takes two inputs from user namely employee id and another is employee income. And then calculates the tax based on tax data provided in tax rates file.
6. For option 2, program takes one argument that is employee id to search employee’s tax details in tax report file.
7. Program will end with option 3 which is to end the program.

# TASK OUTPUT

|  |  |
| --- | --- |
| **File reading and writing, calculating tax, Searching data** | |
| **Test Data** | **Screenshot** |
| **File reading and writing** | |
| Checking file if they exist or not.  If not, then program will create it with specified format. |  |
| Reads and writes data from and to existing file |  |
| **Calculation** | |
| Single entry.  1. Employee id - 1111  Employee income - 102000 |  |
| Multiple entry.  1. Employee id - 2222  Employee income - 42000  2. Employee id - 3333  Employee income - 148000  3. Employee id - 4444  Employee income - 75000 |  |
| Repeated Entry.  1. Employee id - 4444  Employee income - 94000  2. Employee id - 5555  Employee income - 115000 |  |
| **Searching** | |
| Single entry search  1. Employee id – 2222 |  |
| Multiple entry search  1. Employee id – 3333  2. Employee id – 1111 |  |
| Repeated entry search (latest data)  1. Employee id - 4444 |  |
| Data not found  1. Employee id - 6666 |  |

# TASK CODE

**File: TaxManagementSystem.java**

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** TaxManagementSystem {

// defining constants for file path

**public** **static** **final** String ***TAX\_RATE\_FILE*** = "C:\\Users\\Vismay\\eclipse-workspace\\first\_java\_project\\src\\first\_java\_project\\taxrates.txt";

**public** **static** **final** String ***TAX\_REPORT\_FILE*** = "C:\\Users\\Vismay\\eclipse-workspace\\first\_java\_project\\src\\first\_java\_project\\taxreport.txt";

**public** **static** **void** main(String args[]) **throws** Exception

{

System.***out***.println("Welcome to Tax Management System \n");

*fileExists*(); // creates new files if it does not exist

ArrayList<TaxRates> taxData = *ReadTaxRatesFile*(); // creating arraylist to store tax data each as an object form tax rates file

ArrayList<Employee> employeeData = *ReadTaxReportFile*(); // reads data from employee text file and stores all data in array list using read method

*WriteTaxReportFile*(employeeData); // write method called to write employee details

Scanner userInput = **new** Scanner(System.***in***);

**boolean** loop = **true**;

// using while to loop through application until stopped by user

**while**(loop)

{

System.***out***.println("Please select one of the following options: \n1. Calculate tax \n2. Search Tax \n3. Exit");

**int** option = userInput.nextInt();

// using if else statement to check which option has been chosen by user

**if**(option == 1)

{

String yesOrNo = "Y";

// using while loop to check if user wants to perform option 1 again

**while**(yesOrNo.equals("Y"))

{

Employee newEmployee = **new** Employee(); // creating employee object to store employee record

Scanner scan = **new** Scanner(System.***in***); // defining scanner to get user input for employee details

System.***out***.println("Enter employee id: ");

String newEmployeeId = scan.nextLine();

System.***out***.println("Enter employee income: ");

**double** newEmployeeIncome = scan.nextDouble();

// using for loop to set values for employee object with the details passed by user

**for**(**int** i = taxData.size()-1; i >= 0; i--)

{

// checks the slab in which employee income comes

**if**(newEmployeeIncome >= taxData.get(i).getIncomeSlab())

{

// formula to calculate tax amount based on employee income and tax income slab

Double calculated\_tax = taxData.get(i).getFixedAmount() + (((newEmployeeIncome - (taxData.get(i).getIncomeSlab()-1)) \* taxData.get(i).getFixedRate()));

newEmployee.setEmployeeId(newEmployeeId); // sets employee id to employee object

newEmployee.setTaxableIncome(newEmployeeIncome); // sets employee income to employee object

newEmployee.setTaxAmount(calculated\_tax); // calculates tax amount and sets tax amount in employee object

employeeData.add(newEmployee); // adds new employee to employee objects list

*WriteTaxReportFile*(employeeData); // writes data to tax report text file

employeeData = *ReadTaxReportFile*(); // reads tax report text file to update the data

System.***out***.println("Tax for this employee has been calculated and updated in records file.");

**break**;

}

// checks if employee's income is the least and do not pay the tax

**else** **if**(newEmployeeIncome < taxData.get(i).getIncomeSlab() && i == 0)

{

System.***out***.println("You do not have to pay the tax.");

**break**;

}

**else**

{

**continue**;

}

}

System.***out***.println("Do you want to calculate again? Y or N.");

yesOrNo = scan.next().toUpperCase();

// checks user input and validates it

**while**(!yesOrNo.equals("Y") && !yesOrNo.equals("N"))

{

System.***out***.println("Please choose Y or N only.");

System.***out***.println("Do you want to calculate again? Y or N.");

yesOrNo = scan.nextLine().toUpperCase();

}

}

}

// checks if user choose to search tax details

**else** **if**(option == 2)

{

String yesOrNo = "Y";

// using while loop to search the employee details

**while**(yesOrNo.equals("Y"))

{

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter employee id to search employee details.");

String searchEmployeeId = scan.nextLine();

// using reverse for loop to search the latest record for the same employee

**for**(**int** i = employeeData.size()-1; i >= 0; i--)

{

// checks if the user record already exists

**if**(searchEmployeeId.equals(employeeData.get(i).getEmployeeId()))

{

System.***out***.println("Following are the details for employee with id: " + employeeData.get(i).getEmployeeId() + "\nTaxable Income: " + employeeData.get(i).getTaxableIncome() + "\nTax Amount: " + employeeData.get(i).getTaxAmount() + ". \n");

**break**;

}

// reports back if no employee with that details exists

**else** **if**(!searchEmployeeId.equals(employeeData.get(i).getEmployeeId()) && i == 0)

{

System.***out***.print("No employee with this id found. Try again with different id. \n");

}

}

System.***out***.println("Do you want to search again? Y or N.");

yesOrNo = scan.nextLine().toUpperCase();

// checks user input and validates it

**while**(!yesOrNo.equals("Y") && !yesOrNo.equals("N"))

{

System.***out***.println("Please choose Y or N only.");

System.***out***.println("Do you want to search again? Y or N.");

yesOrNo = scan.nextLine().toUpperCase();

}

}

}

// checks if user wants to exit through main loop to exit the system

**else** **if**(option == 3)

{

loop = **false**;

System.***out***.println("This system will exit now. Bye.");

**break**;

}

}

}

// defining a method to check if files exists or not

**public** **static** **void** fileExists() **throws** Exception

{

// open text report file

File taxReportFile = **new** File(***TAX\_REPORT\_FILE***);

// opens tax rates file

File taxRatesFile = **new** File(***TAX\_RATE\_FILE***);

// checks if tax rates file does not exist then creates a new file

**if**(!taxRatesFile.exists())

{

System.***out***.println("Tax rates file does not exist. New file with the data has been created. \n");

// creates a new tax rate file

*createTaxRateFile*();

}

// checks if tax report file does not exist then creates a new file

**if**(!taxReportFile.exists())

{

System.***out***.println("Tax report file does not exist. New file with the data has been created. There are no data in the file currently. Create new employee records and then use search function. \n");

// creates a new tax report file

*CreateTaxReportFile*();

}

}

// creates new tax rates file if does not exist

**public** **static** **void** createTaxRateFile() **throws** Exception

{

// creates a new text file to write data

FileWriter createTaxRateFile = **new** FileWriter(***TAX\_RATE\_FILE***);

createTaxRateFile.write(String.*format*("%-30s %-50s \n","Taxable Income", "Tax on income")); // writes data in specific format for consistency

createTaxRateFile.write(String.*format*("%-30s %-50s \n","0 - $18,200", "0"));

createTaxRateFile.write(String.*format*("%-30s %-50s \n","$18,201 - $37,000", "19c for each $1 over $18,200"));

createTaxRateFile.write(String.*format*("%-30s %-50s \n","$37,001 - $87,000", "$3,572 plus 32.5c for each $1 over $37,000"));

createTaxRateFile.write(String.*format*("%-30s %-50s \n","$87,001 - $180,000", "$19,822 plus 37c for each $1 over $87,000"));

createTaxRateFile.write(String.*format*("%-30s %-50s \n","$180,001 and over", "$54,232 plus 45c for each $1 over $180,000"));

createTaxRateFile.close();

}

// defining method to read tax rates file and store in the object

**public** **static** ArrayList<TaxRates> ReadTaxRatesFile() **throws** Exception

{

// declaring array list to store tax rates objects

ArrayList<TaxRates> taxRatesList = **new** ArrayList<TaxRates>();

// opens tax rates file

File taxRatesFile = **new** File(***TAX\_RATE\_FILE***);

Scanner scanTaxRates = **new** Scanner(taxRatesFile); // reads tax rates file

scanTaxRates.nextLine(); // reads first line to avoid in loop

scanTaxRates.nextLine(); // reads second line to avoid in loop

// using while loop for each and every line in text file

**while**(scanTaxRates.hasNext())

{

TaxRates taxRates = **new** TaxRates(); // creating tax rate object

String[] taxRatesDetails = ((scanTaxRates.nextLine()).trim()).split("\\s+"); // splits line by space

taxRates.setIncomeSlab(Double.*parseDouble*(taxRatesDetails[0].replaceAll("[^0-9]", ""))); // gets the first value in line and sets it as income slab in object

// using for loop through all the data to find digit + c format where c stands for cents to get tax rate and gets tax amount

**for**(**int** i = 0; i < taxRatesDetails.length; i++)

{

// checks the condition of digit + c

**if**(taxRatesDetails[i].matches("[cC0-9.]\*"))

{

// if found then, value is stored as tax rate

taxRates.setFixedRate(Double.*parseDouble*(taxRatesDetails[i].replaceAll("[^\\d.]", ""))/100);

}

// checks if the length of line is greater then 10 to get the values of fixed tax amount

**if**(taxRatesDetails.length >= 10)

{

// gets and stores fixed tax amount in object

taxRates.setFixedAmount(Double.*parseDouble*(taxRatesDetails[3].replaceAll("[^0-9]", "")));

}

}

// adds this object to object array list

taxRatesList.add(taxRates);

}

scanTaxRates.close();

**return** taxRatesList;

}

// creates new tax report file if it does not exist

**public** **static** **void** CreateTaxReportFile() **throws** Exception

{

// creates a new text file to write data

FileWriter createTaxReportFile = **new** FileWriter(***TAX\_REPORT\_FILE***);

createTaxReportFile.write(String.*format*("%-15s %-20s %-15s","Employee\_id", "Taxable\_income", "Tax")); // writes data in specific format for consistency

createTaxReportFile.write("\n");

createTaxReportFile.close();

}

// defining method to read employee details from tax report file

**public** **static** ArrayList<Employee> ReadTaxReportFile() **throws** Exception

{

// open text file to read

File taxReportFile = **new** File(***TAX\_REPORT\_FILE***);

ArrayList<Employee> employeeList = **new** ArrayList<Employee>(); // creates array list to store objects of data for each record in file

// checks if files exists or not and performs function accordingly

Scanner scanTaxReport = **new** Scanner(taxReportFile); // reads data from text file

scanTaxReport.nextLine(); // reads first heading line to avoid it in loop

// using while loop through all the data in text file to store in object

**while**(scanTaxReport.hasNext())

{

Employee employee = **new** Employee(); // creates employee object

String[] employeeDetails = ((scanTaxReport.nextLine()).trim()).split("\\s+"); // splits line with space

employee.setEmployeeId(employeeDetails[0]); // stores first data as employee id

employee.setTaxableIncome(Double.*parseDouble*(employeeDetails[1])); // stores second data as employee income

employee.setTaxAmount((Double.*parseDouble*(employeeDetails[2]))); // stores third data as calculated tax amount

employeeList.add(employee); // adds object to object array list

}

scanTaxReport.close();

**return** employeeList;

}

// defining method to write employee details to tax report text file

**public** **static** **void** WriteTaxReportFile(ArrayList<Employee> employeeDetails) **throws** Exception

{

// opens text file to write data

FileWriter writer = **new** FileWriter(***TAX\_REPORT\_FILE***);

writer.write(String.*format*("%-15s %-20s %-15s","Employee\_id", "Taxable\_income", "Tax")); // writes data in specific format for consistency

writer.write("\n");

// using for loop to get individual record data to write in file

**for**(**int** i = 0; i < employeeDetails.size(); i++)

{

// stores each record in string in specific format

String final\_str = String.*format*("%-15s %-20.2f %-15.2f", employeeDetails.get(i).getEmployeeId(), employeeDetails.get(i).getTaxableIncome(), employeeDetails.get(i).getTaxAmount());

writer.write(final\_str); // writes string to text file

writer.write("\n");

}

writer.close();

}

}

**File: Employee.java**

**public** **class** Employee {

// declaring all the variables to store values for employee details

**private** String employeeId;

**private** **double** taxableIncome;

**private** **double** taxAmount;

// no args constuctor

**public** Employee()

{

}

// get method to get employee id

**public** String getEmployeeId()

{

**return** employeeId;

}

// set method to set new employee id

**public** **void** setEmployeeId(String newEmployeeId)

{

employeeId = newEmployeeId;

}

// get method to get taxable income

**public** **double** getTaxableIncome()

{

**return** taxableIncome;

}

// set method to set new taxable income

**public** **void** setTaxableIncome(**double** newTaxableIncome)

{

taxableIncome = newTaxableIncome;

}

// get method to get tax amount

**public** **double** getTaxAmount()

{

**return** taxAmount;

}

// set method to set new tax amount

**public** **void** setTaxAmount(**double** newTaxAmount)

{

taxAmount = newTaxAmount;

}

}

**File:TaxRates.java**

**public** **class** TaxRates {

// declaring all the variables that relates to important factors needed for tax calculation

**private** **double** incomeSlab;

**private** **double** fixedAmount;

**private** **double** fixedRate;

// constructor of the class

**public** TaxRates()

{

}

// get method to get income slab value

**public** **double** getIncomeSlab()

{

**return** incomeSlab;

}

// set method to set income slab value to new value

**public** **void** setIncomeSlab(**double** newIncomeSlab)

{

incomeSlab = newIncomeSlab;

}

// get method to get fixed amount value

**public** **double** getFixedAmount()

{

**return** fixedAmount;

}

// set method to set fixed amount value to new value

**public** **void** setFixedAmount(**double** newFixedAmount)

{

fixedAmount = newFixedAmount;

}

// get method to get fixed rate value

**public** **double** getFixedRate()

{

**return** fixedRate;

}

// set method to set fixed rate value to new value

**public** **void** setFixedRate(**double** newFixedRate)

{

fixedRate = newFixedRate;

}

}